

Debugging

CS 113: Introduction to C

Instructor: Saikat Guha

Cornell University

Spring 2007, Lecture 5

Print Debugging

- ▶ Manually insert debugging statements
- ▶ Debugging statements print to screen
 - ▶ Caution: `stdout` is buffered. `printf` output may not appear before program crashes.
 - ▶ Solution: `stderr` is unbuffered.

printf debugging

```
fprintf(stderr, "%d %p", i, p);
```

- ▶ `%d` – int
- ▶ `%s` – char *
- ▶ `%p` – any pointer
- ▶ see man page for others `$ man 3 printf`

debug.c: Trace Information

```
#include <stdio.h>

int main(int argc, char **argv) {
    fprintf(stderr, "%s:%d:%s\t%s\n", __FILE__,
            __LINE__, __FUNCTION__, argv[0]);

    fprintf(stderr, "%s:%d:%s\t%s\n", __FILE__,
            __LINE__, __FUNCTION__, argv[1]);

    fprintf(stderr, "%s:%d:%s\t%s\n", __FILE__,
            __LINE__, __FUNCTION__, argv[2]);
}
```

```
debug.c:5:main ./test
debug.c:8:main hello
debug.c:11:main world
```

GDB: GNU Debugger

```
[saikat@submit cs113]$ gcc -g -o cmd cmd.c
[saikat@submit cs113]$ ./cmd foo
Segmentation fault
[saikat@submit cs113]$ gdb ./cmd
...
(gdb) b main
Breakpoint 1 at 0x80483a4: file cmd.c, line 3.
(gdb) r foo
...
Breakpoint 1, main (argc=1209306428, argv=0x4802f4c6) at
cmd.c:3
3 int main(int argc, char **argv) {
(gdb) n
main (argc=2, argv=0xbfb646e4) at cmd.c:6
6 n = atoi(argv[1]);
(gdb) p argc
$1 = 2
```

GDB: GNU Debugger

```
(gdb) p argv[0]  
$2 = 0xbfb65c84 "/home/netid/cs113/cmd"
```

```
(gdb) c  
Continuing.
```

```
Program received signal SIGSEGV, Segmentation fault.  
0x48045eae in ___strtol_l_internal () from /lib/libc.so.6
```

```
(gdb) bt  
#0 0x48045eae in ___strtol_l_internal () from  
/lib/libc.so.6  
#1 0x48045c57 in __strtol_internal () from /lib/libc.so.6  
#2 0x48043511 in atoi () from /lib/libc.so.6  
#3 0x080483eb in main (argc=2, argv=0xbfb646e4) at cmd.c:7
```

```
(gdb) fr 3  
#3 0x080483eb in main (argc=2, argv=0xbfb646e4) at cmd.c:7  
7 m = atoi(argv[2]);
```

```
(gdb) p argv[2]  
$3 = 0x0
```

GDB: Commands

- ▶ **b** `<function>` – Breakpoint on entering function
- ▶ **r** `<args>` – Run program
- ▶ **list** – print C code
- ▶ **n** – execute one statement
- ▶ **s** – execute one step (step into function calls)
- ▶ **c** – Continue running program
- ▶ **p** `<variable>` – print the value of a variable
- ▶ **bt** – Backtrace the stack
- ▶ **fr** `<num>` – Make stackframe `<num>` current frame for printing variables
- ▶ **q** – Quit
- ▶ **help** – More GDB help

Things to try

- ▶ Crash a program by dereferencing a NULL pointer.
- ▶ Crash a program by running out of stack space.
- ▶ Crash a program by clobbering the stack (e.g. the return address).
- ▶ Crash a program by calling `abort()`.

... debug each of these cases using GDB